LabB Cholesky Factorization

Speaker: Hua-Yang Weng   Date: 2021.10.31

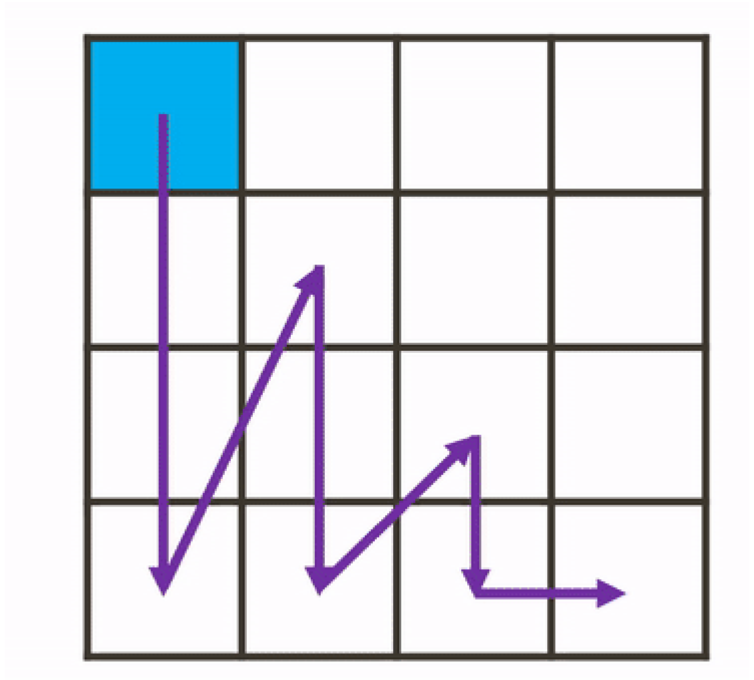
Github: https://github.com/Yuoto/LabB_cholesky_vitis

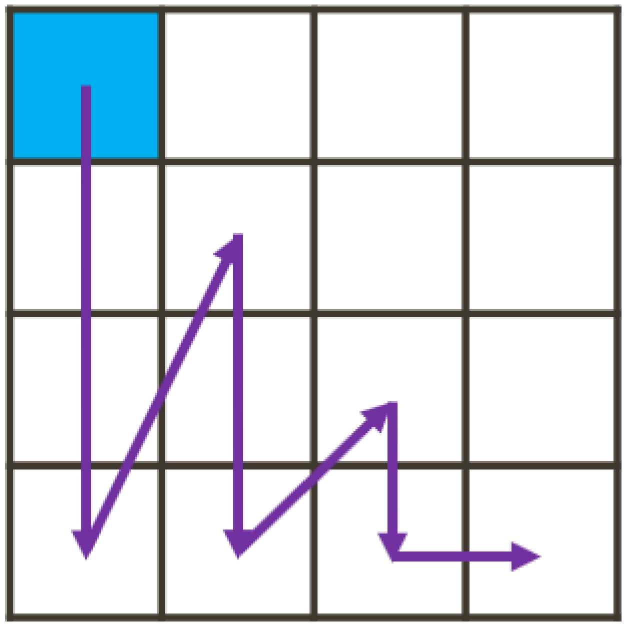
Take home Idea

- Cholesky Algorithm Flow
- Vitis HLS
  - 1. Pipeline
  - 2. Pipeline – II=1
  - 3. Code Refactor – Unroll
  - 4. DataType
- Vitis System
  - Compare with CPU
- Appendix
  - Code Refactor – Dataflow

Cholesky Flow

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathrm{T}} = \begin{pmatrix} \mathbf{L}_{11} & 0 & 0 \\ \mathbf{L}_{21} & \mathbf{L}_{22} & 0 \\ \mathbf{L}_{31} & \mathbf{L}_{32} & \mathbf{L}_{33} \end{pmatrix} \begin{pmatrix} \mathbf{L}_{11} & \mathbf{L}_{21} & \mathbf{L}_{31} \\ 0 & \mathbf{L}_{22} & \mathbf{L}_{32} \\ 0 & 0 & \mathbf{L}_{33} \end{pmatrix}$$
$$= \begin{pmatrix} \mathbf{L}_{11}^2 & & (\text{symmetric}) \\ \mathbf{L}_{21}\mathbf{L}_{11} & \mathbf{L}_{21}^2 + \mathbf{L}_{22}^2 & \\ \mathbf{L}_{31}\mathbf{L}_{11} & \mathbf{L}_{31}\mathbf{L}_{21} + \mathbf{L}_{32}\mathbf{L}_{22} & \mathbf{L}_{31}^2 + \mathbf{L}_{32}^2 + \mathbf{L}_{33}^2 \end{pmatrix}$$





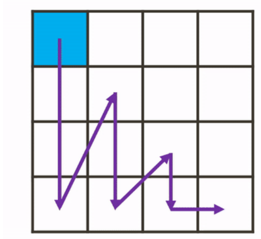
$$\mathbf{L}_{j,j} = \sqrt{A_{j,j} - \sum_{k=1}^{j-1} \mathbf{L}_{j,k}^2}$$

$$\mathbf{L}_{i,j} = \frac{1}{\mathbf{L}_{j,j}} \left(A_{i,j} - \sum_{k=1}^{j-1} \mathbf{L}_{i,k}\mathbf{L}_{j,k} \right), \qquad \text{for } i > j$$

Insights:
- For each column data, they can be computed in parallel
- The order of column cannot be changed

Vitis HLS

1. No pragma (pipeline)



```
extern "C" void cholesky_kernel(int diagSize, double* matrixA) {

   dataType dataA[MAXN][MAXN];
   diagSize = 512;

   for(int i = 0; i < diagSize; i++){
      for (int j = 0; j < diagSize; j++) {
      dataA[i][j] = matrixA[i*diagSize + j];
      }
   }
```

Read Data
262144 cycles, II=1, Iter. Lat=2

```
  double tmp1=sqrt(dataA[0][0]);

  dataA[0][0] = tmp1;
  Loop_first_col:
  for (int i = 1; i < diagSize; i++){
     dataA[i][0] = dataA[i][0]/tmp1;
  }
```

Calculate First Column
571 cycles, II=1, Iter. Lat=62
Q1: partition in dim1?

```
Loop_col:
for (int j = 1; j < diagSize; ++j){
   dataType tmp = 0;

   Loop_diag:
   for(int k = 0; k < j; k++){
      tmp += dataA[j][k]*dataA[j][k];
   }

   dataA[j][j] = sqrt(dataA[j][j] - tmp);

   if (j < diagSize - 1){
   Loop_row:
   for(int i = j+1; i < diagSize; ++i){
      dataType tmp2=0;
      Loop_vec_mul:
      for(int k = 0; k < j; k++){
         tmp2 += dataA[i][k]*dataA[j][k];
      }
      dataA[i][j] = (dataA[i][j] - tmp2)/dataA[j][j];
   }
   }
}
```

Loop_diag
Trip C = 1~511,
II=6, Iter. Lat=16

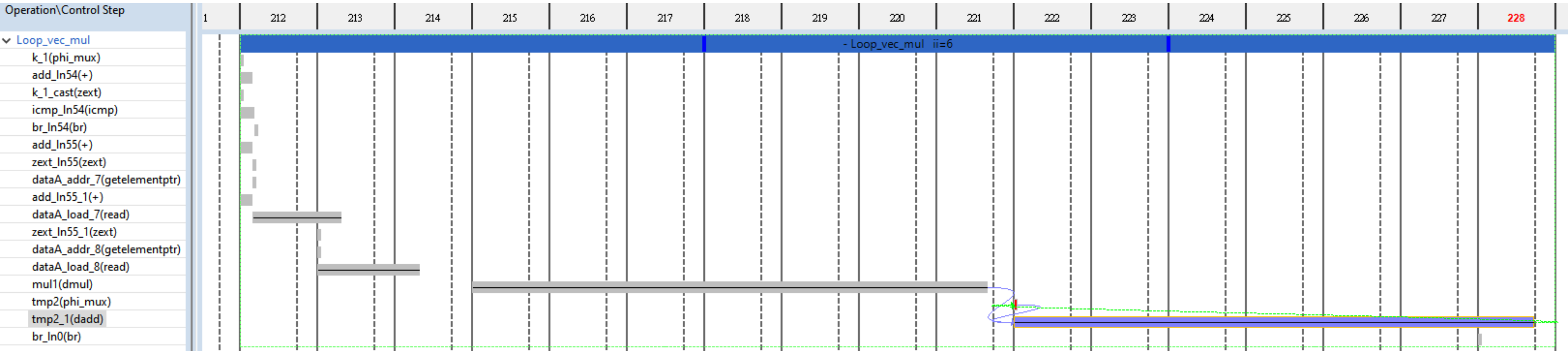
Loop_vec_mul
Trip C = 1~511,
II=6,Iter. Lat=17

Write Data
262145 cycles, II=1, Iter. Lat=3

```
 for (int i = 0; i < diagSize; i++) {
    for (int j = 0; j < diagSize; j++) {
       matrixA[i * diagSize + j] = dataA[i][j];
    }
 }
```

II Violation

- ***tmp2+=*** inner-product **data dependency**
- Loop_vec_mul: IL=17 (3 read +7 mul + 7 add), II=6

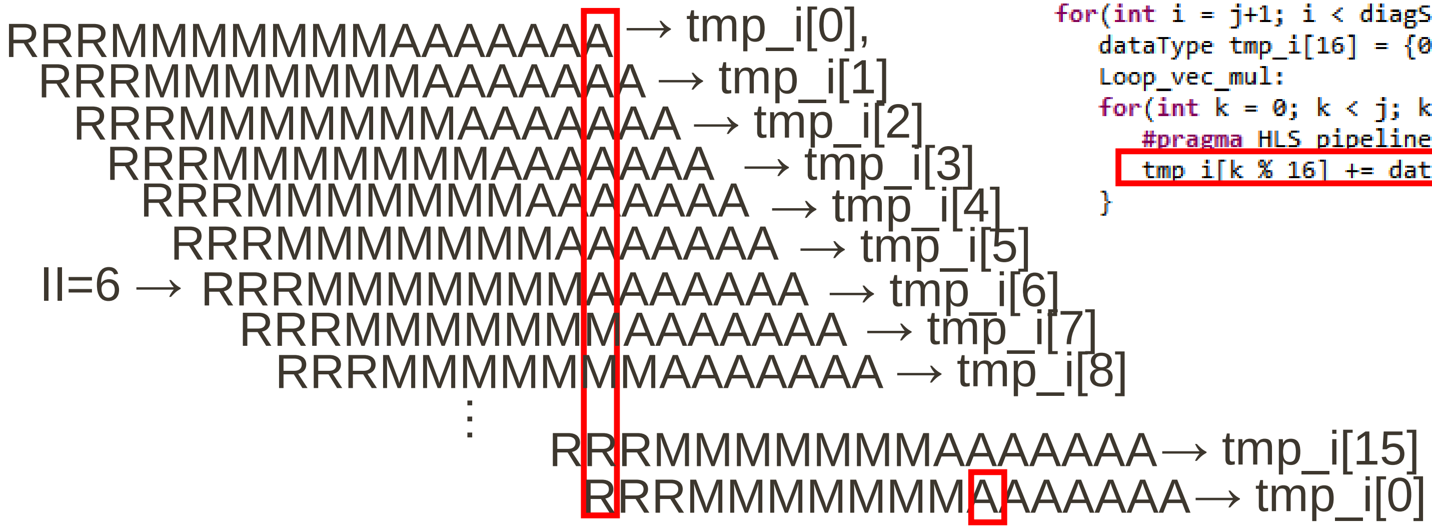


```
RRRMMMMMMMAAAAAAA
  RRRMMMMMMMAAAAAAA
    RRRMMMMMMMAAAAAAA
      RRRMMMMMMMAAAAAAA
        RRRMMMMMMMAAAAAAA
          RRRMMMMMMMAAAAAAA
II=6 → RRRMMMMMMMAAAAAAA
```

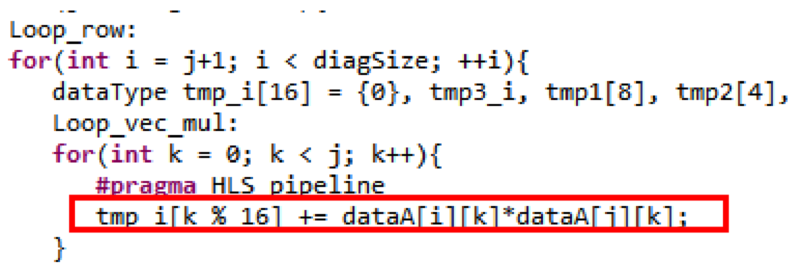
```
if (j < diagSize - 1){
Loop_row:
for(int i = j+1; i < diagSize; ++i){
   dataType tmp2=0;
   Loop_vec_mul:
   for(int k = 0; k < j; k++){
      tmp2 += dataA[i][k]*dataA[j][k];
   }
   dataA[i][j] = (dataA[i][j] - tmp2)/dataA[j][j];
}
}
}
```

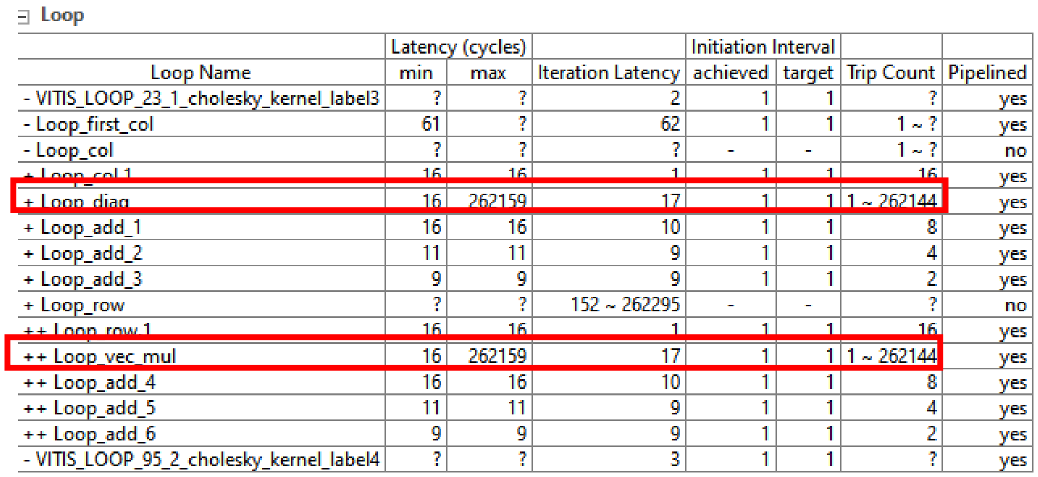
2. Pipeline – II=1

- Break the ***tmp+=*** inner-product **data dependency**
- Add buffer (depth > II)



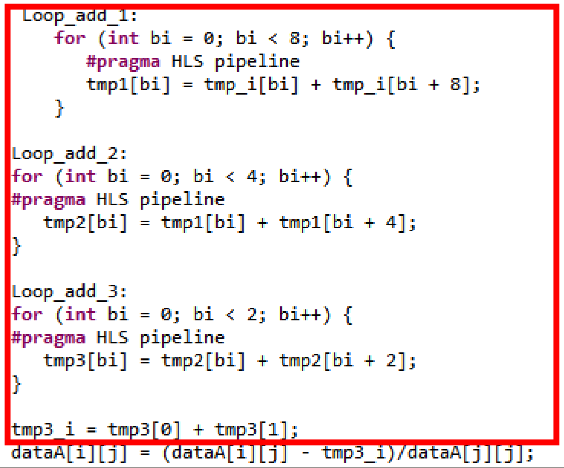
RRRMMMMMMMAAAAAAA → tmp_i[0],
RRRMMMMMMMAAAAAAA → tmp_i[1]
RRRMMMMMMMAAAAAAA → tmp_i[2]
RRRMMMMMMMAAAAAAA → tmp_i[3]
RRRMMMMMMMAAAAAAA → tmp_i[4]
RRRMMMMMMMAAAAAAA → tmp_i[5]
II=6 → RRRMMMMMMMAAAAAAA → tmp_i[6]
RRRMMMMMMMAAAAAAA → tmp_i[7]
RRRMMMMMMMAAAAAAA → tmp_i[8]
⋮
RRRMMMMMMMAAAAAAA→ tmp_i[15]
RRRMMMMMMMAAAAAAA→ tmp_i[0]


```
Loop_row:
for(int i = j+1; i < diagSize; ++i){
    dataType tmp_i[16] = {0}, tmp3_i, tmp1[8], tmp2[4],
    Loop_vec_mul:
    for(int k = 0; k < j; k++){
        #pragma HLS pipeline
        tmp_i[k % 16] += dataA[i][k]*dataA[j][k];
    }
```



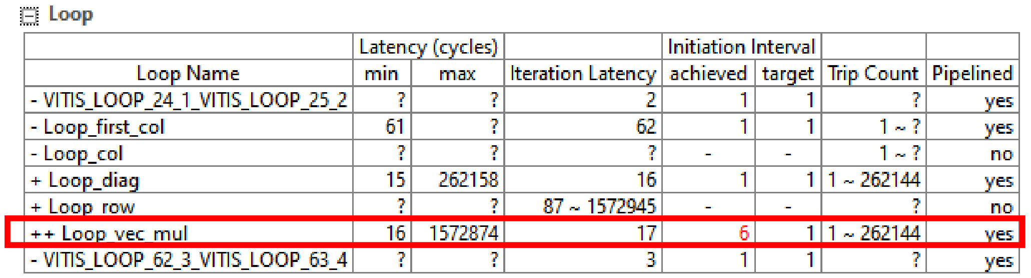
Loop

Loop Name	Latency (cycles)		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- VITIS_LOOP_23_1_cholesky_kernel_label3	?	?	2	1	1	?	yes
- Loop_first_col	61	?	62	1	1	1 ~ ?	yes
- Loop_col	?	?	?	-	-	1 ~ ?	no
+ Loop_col_1	16	16	1	1	1	16	yes
+ Loop_diag	16	262159	17	1	1	1 ~ 262144	yes
+ Loop_add_1	16	16	10	1	1	8	yes
+ Loop_add_2	11	11	9	1	1	4	yes
+ Loop_add_3	9	9	9	1	1	2	yes
+ Loop_row	?	?	152 ~ 262295	-	-	?	no
++ Loop_row_1	16	16	1	1	1	16	yes
++ Loop_vec_mul	16	262159	17	1	1	1 ~ 262144	yes
++ Loop_add_4	16	16	10	1	1	8	yes
++ Loop_add_5	11	11	9	1	1	4	yes
++ Loop_add_6	9	9	9	1	1	2	yes
- VITIS_LOOP_95_2_cholesky_kernel_label4	?	?	3	1	1	?	yes


```
    Loop_add_1:
        for (int bi = 0; bi < 8; bi++) {
            #pragma HLS pipeline
            tmp1[bi] = tmp_i[bi] + tmp_i[bi + 8];
        }

Loop_add_2:
for (int bi = 0; bi < 4; bi++) {
#pragma HLS pipeline
    tmp2[bi] = tmp1[bi] + tmp1[bi + 4];
}

Loop_add_3:
for (int bi = 0; bi < 2; bi++) {
#pragma HLS pipeline
    tmp3[bi] = tmp2[bi] + tmp2[bi + 2];
}

tmp3_i = tmp3[0] + tmp3[1];
dataA[i][j] = (dataA[i][j] - tmp3_i)/dataA[j][j];
```

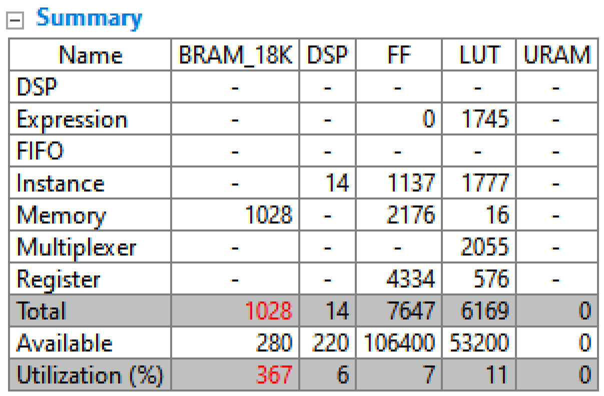
Compare Report (512x512)

- II=6



Loop

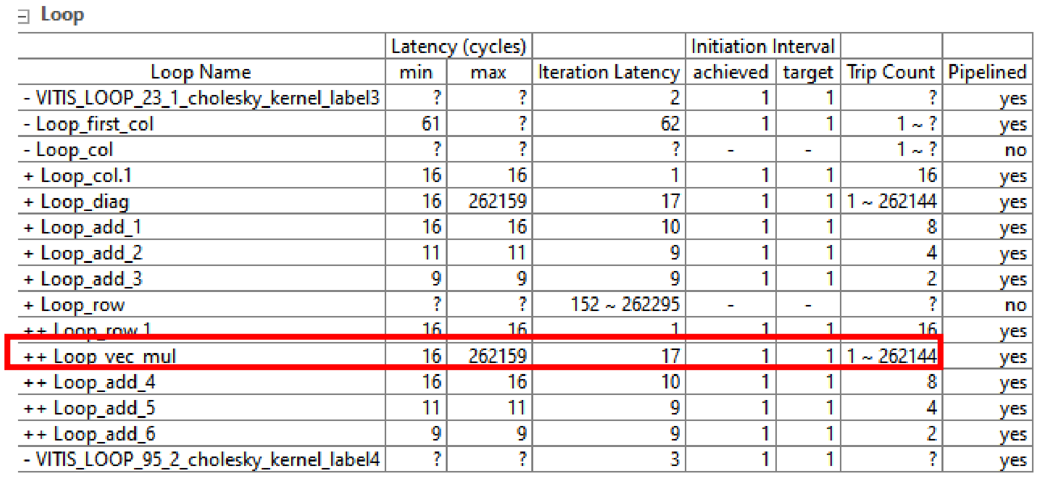
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- VITIS_LOOP_24_1_VITIS_LOOP_25_2	?	?	2	1	1	?	yes
- Loop_first_col	61	?	62	1	1	1 ~ ?	yes
- Loop_col	?	?	?	-	-	1 ~ ?	no
+ Loop_diag	15	262158	16	1	1	1 ~ 262144	yes
+ Loop_row	?	?	87 ~ 1572945	-	-	?	no
++ Loop_vec_mul	16	1572874	17	6	1	1 ~ 262144	yes
- VITIS_LOOP_62_3_VITIS_LOOP_63_4	?	?	3	1	1	?	yes



Summary

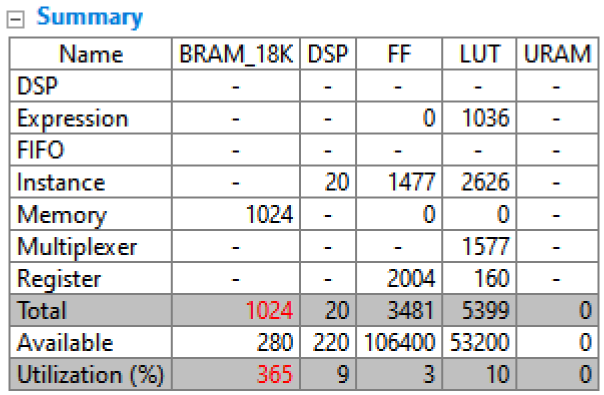
Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1745	-
FIFO	-	-	-	-	-
Instance	-	14	1137	1777	-
Memory	1028	-	2176	16	-
Multiplexer	-	-	-	2055	-
Register	-	-	4334	576	-
Total	1028	14	7647	6169	0
Available	280	220	106400	53200	0
Utilization (%)	367	6	7	11	0

- II=1



Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- VITIS_LOOP_23_1_cholesky_kernel_label3	?	?	2	1	1	?	yes
- Loop_first_col	61	?	62	1	1	1 ~ ?	yes
- Loop_col	?	?	?	-	-	1 ~ ?	no
+ Loop_col.1	16	16	1	1	1	16	yes
+ Loop_diag	16	262159	17	1	1	1 ~ 262144	yes
+ Loop_add_1	16	16	10	1	1	8	yes
+ Loop_add_2	11	11	9	1	1	4	yes
+ Loop_add_3	9	9	9	1	1	2	yes
+ Loop_row	?	?	152 ~ 262295	-	-	?	no
++ Loop_row.1	16	16	1	1	1	16	yes
++ Loop_vec_mul	16	262159	17	1	1	1 ~ 262144	yes
++ Loop_add_4	16	16	10	1	1	8	yes
++ Loop_add_5	11	11	9	1	1	4	yes
++ Loop_add_6	9	9	9	1	1	2	yes
- VITIS_LOOP_95_2_cholesky_kernel_label4	?	?	3	1	1	?	yes

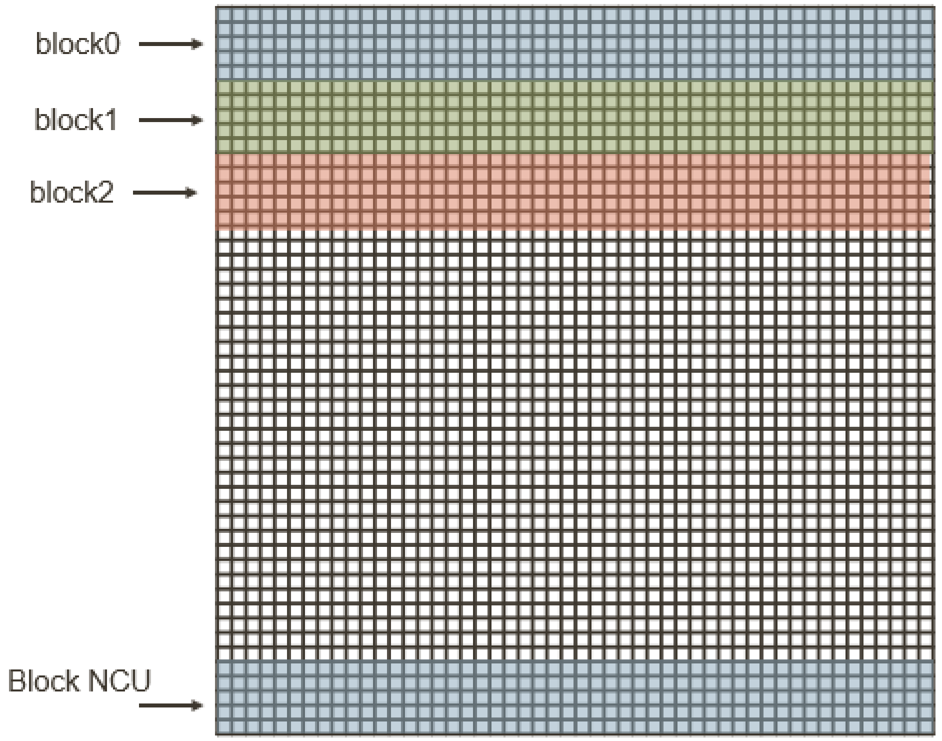


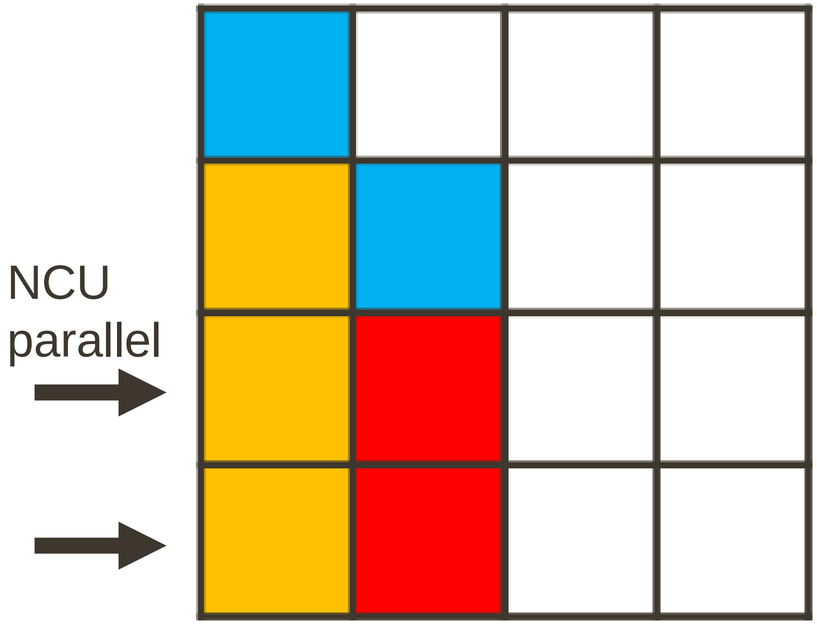
Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1036	-
FIFO	-	-	-	-	-
Instance	-	20	1477	2626	-
Memory	1024	-	0	0	-
Multiplexer	-	-	-	1577	-
Register	-	-	2004	160	-
Total	1024	20	3481	5399	0
Available	280	220	106400	53200	0
Utilization (%)	365	9	3	10	0

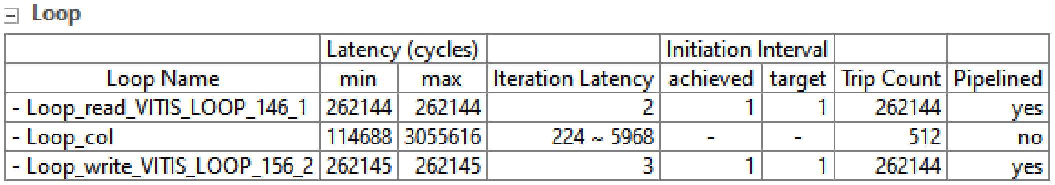
3. Code Refactor – Unroll

- Split **tiles(blocks)**
- Perform **pipelined** inner-product **in parallel**

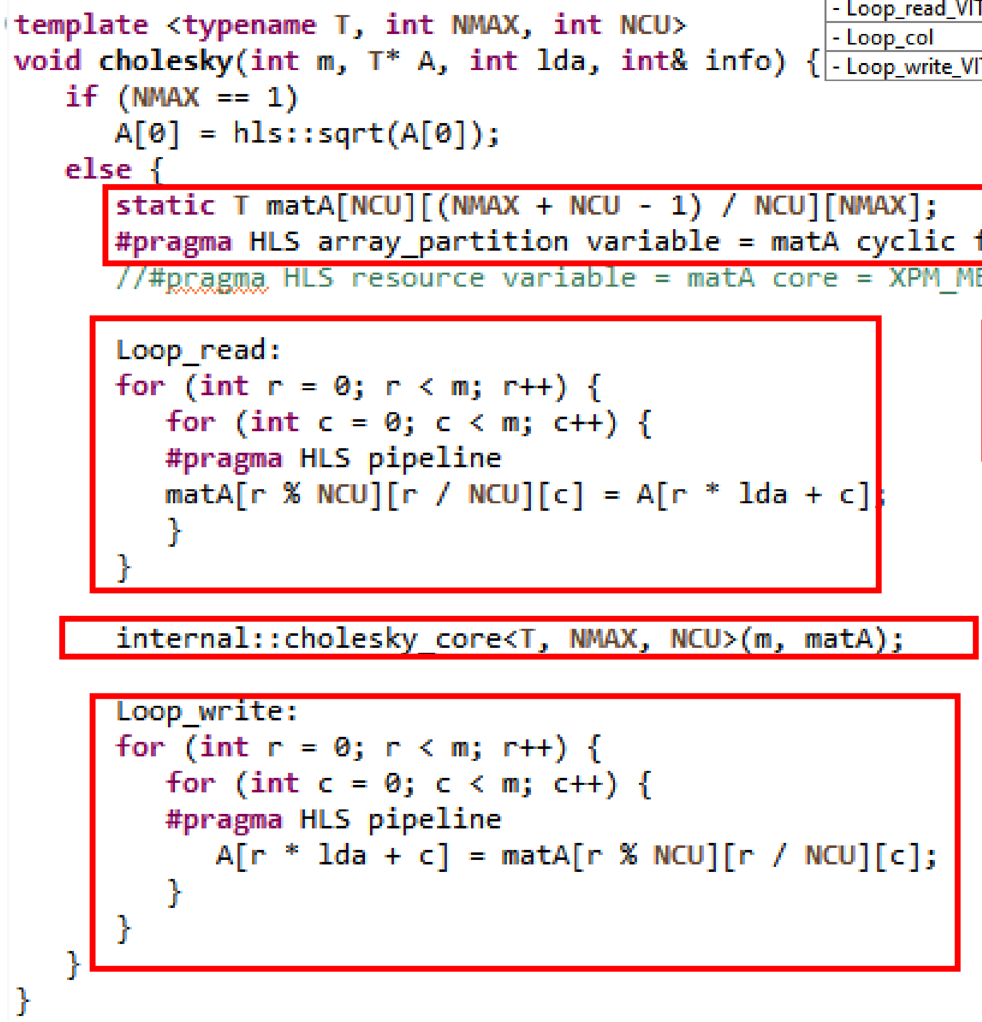




3. Code Refactor – Unroll


Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop_read_VITIS_LOOP_146_1	262144	262144	2	1	1	262144	yes
- Loop_col	114688	3055616	224 ~ 5968	-	-	512	no
- Loop_write_VITIS_LOOP_156_2	262145	262145	3	1	1	262144	yes


```
template <typename T, int NMAX, int NCU>
void cholesky(int m, T* A, int lda, int& info) {
   if (NMAX == 1)
      A[0] = hls::sqrt(A[0]);
   else {
      static T matA[NCU][(NMAX + NCU - 1) / NCU][NMAX];
      #pragma HLS array_partition variable = matA cyclic factor = NCU dim = 1
      //#pragma HLS resource variable = matA core = XPM_MEMORY uram

      Loop_read:
      for (int r = 0; r < m; r++) {
         for (int c = 0; c < m; c++) {
         #pragma HLS pipeline
         matA[r % NCU][r / NCU][c] = A[r * lda + c];
         }
      }

      internal::cholesky_core<T, NMAX, NCU>(m, matA);

      Loop_write:
      for (int r = 0; r < m; r++) {
         for (int c = 0; c < m; c++) {
         #pragma HLS pipeline
            A[r * lda + c] = matA[r % NCU][r / NCU][c];
         }
      }
   }
}
```

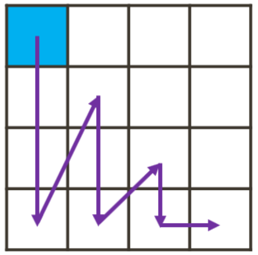
Array-partitioned Matrix buffer

Read Data
262144 cycles, II=1, Iter. Lat=2

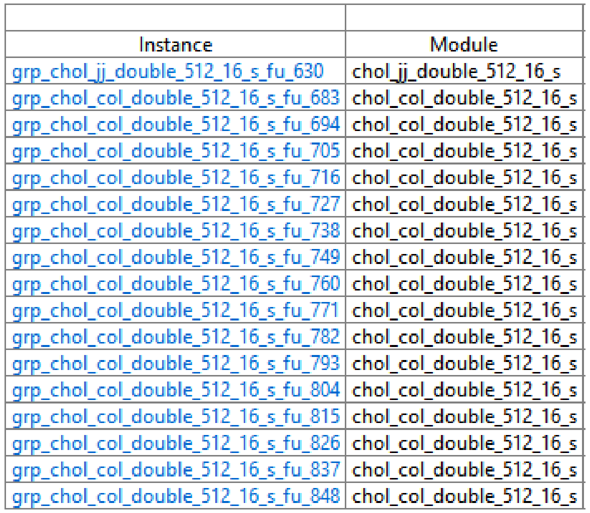
Core
114688~ 3055616 cycles

Write Data
262144 cycles, II=1, Iter. Lat=3

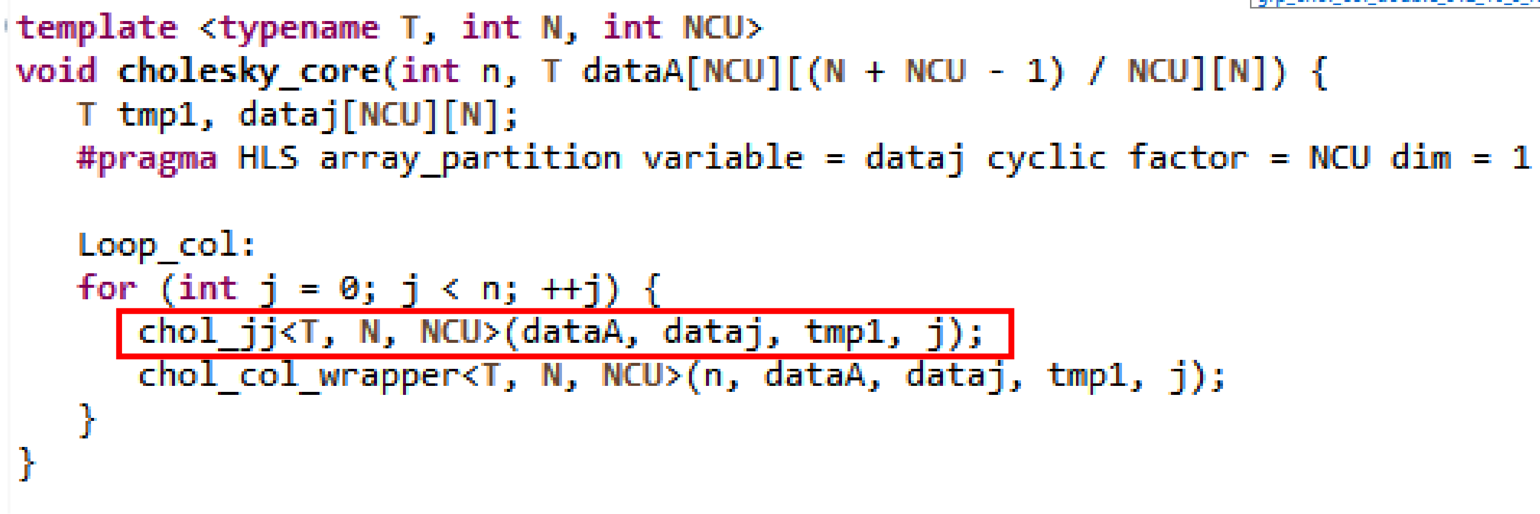
3. Code Refactor – Unroll



- Core: A big **non-pipelined** loop
- For each col j in matrix
  - Calculate diagonal term first (chol_jj unit)
  - Then, **parallel** compute **each rows** with factor **NCU** (chol_col_wrapper unit)

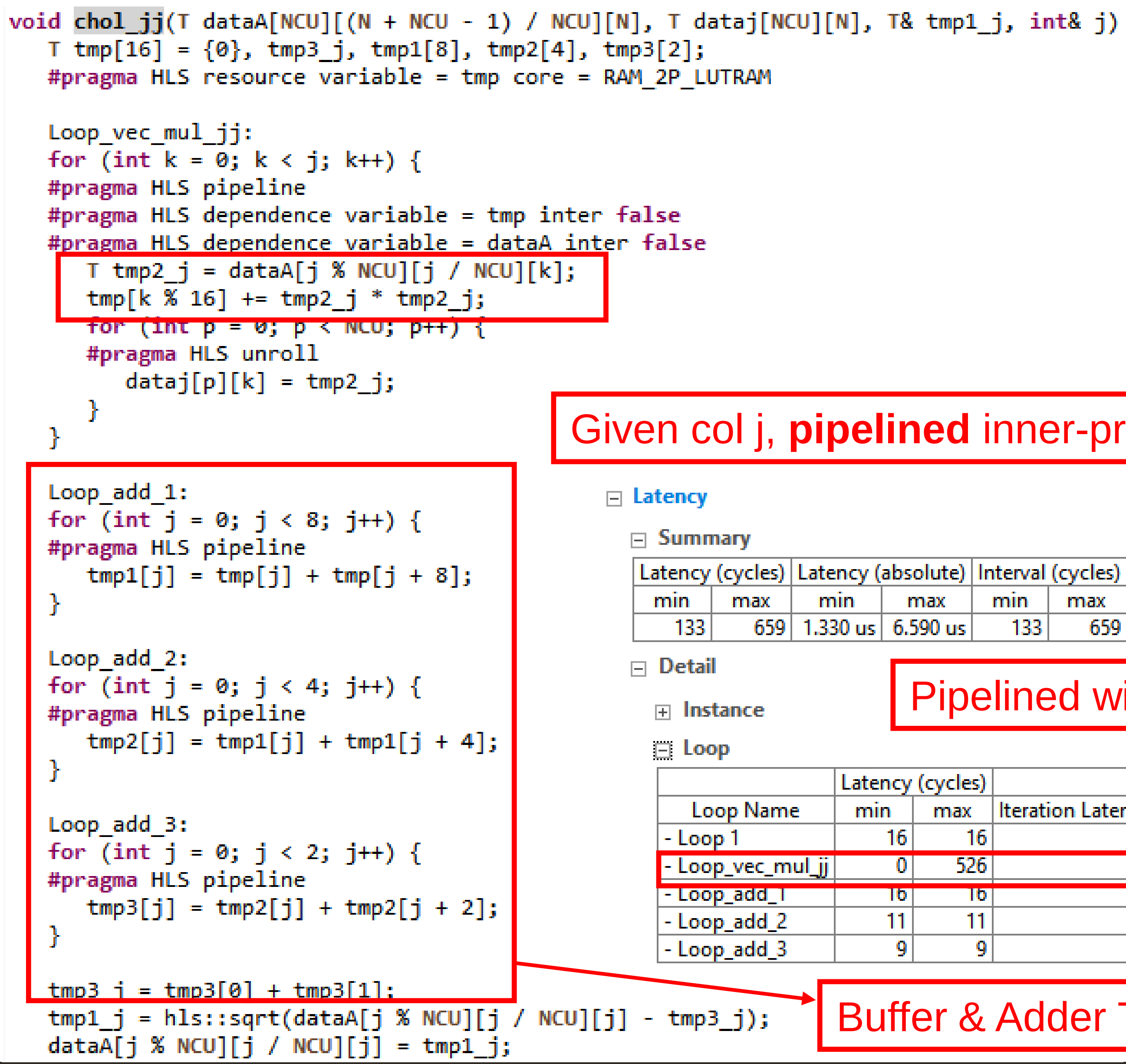


Instance	Module
grp_chol_jj_double_512_16_s_fu_630	chol_jj_double_512_16_s
grp_chol_col_double_512_16_s_fu_683	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_694	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_705	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_716	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_727	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_738	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_749	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_760	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_771	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_782	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_793	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_804	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_815	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_826	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_837	chol_col_double_512_16_s
grp_chol_col_double_512_16_s_fu_848	chol_col_double_512_16_s

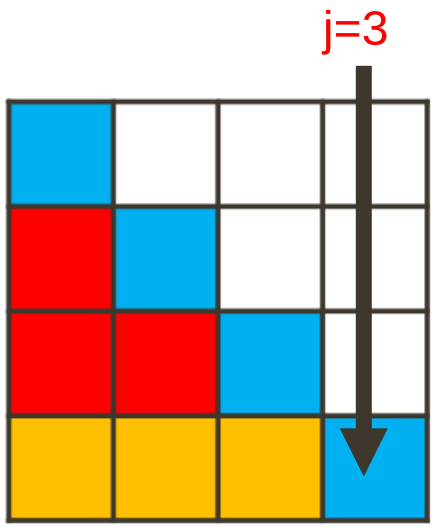


```
template <typename T, int N, int NCU>
void cholesky_core(int n, T dataA[NCU][(N + NCU - 1) / NCU][N]) {
    T tmp1, dataj[NCU][N];
    #pragma HLS array_partition variable = dataj cyclic factor = NCU dim = 1

    Loop_col:
    for (int j = 0; j < n; ++j) {
        chol_jj<T, N, NCU>(dataA, dataj, tmp1, j);
        chol_col_wrapper<T, N, NCU>(n, dataA, dataj, tmp1, j);
    }
}
```

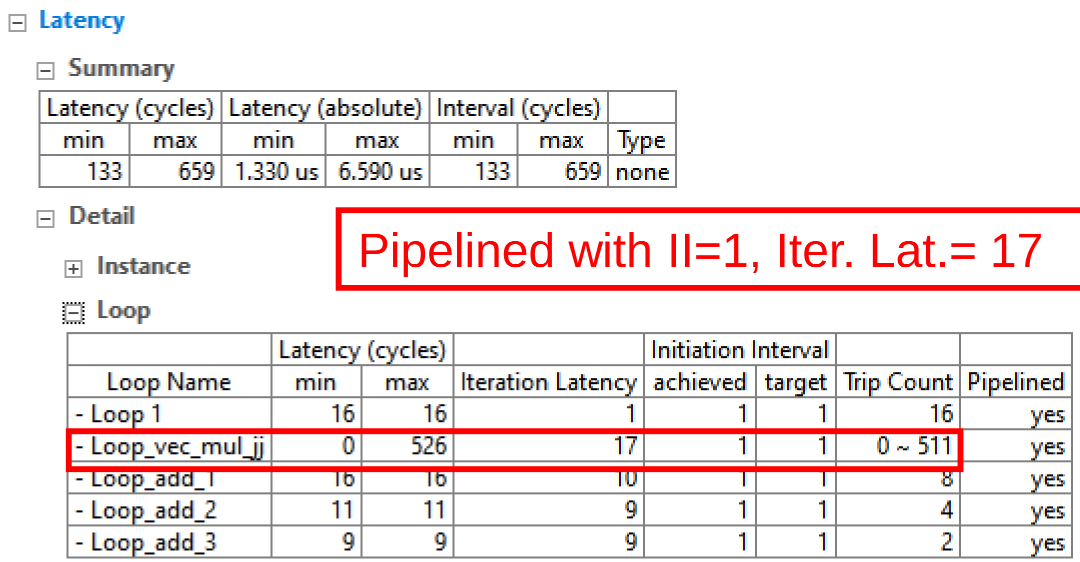
chol_ jj unit



```
void chol_jj(T dataA[NCU][(N + NCU - 1) / NCU][N], T dataj[NCU][N], T& tmp1_j, int& j)
    T tmp[16] = {0}, tmp3_j, tmp1[8], tmp2[4], tmp3[2];
    #pragma HLS resource variable = tmp core = RAM_2P_LUTRAM

    Loop_vec_mul_jj:
    for (int k = 0; k < j; k++) {
    #pragma HLS pipeline
    #pragma HLS dependence variable = tmp inter false
    #pragma HLS dependence variable = dataA inter false
        T tmp2_j = dataA[j % NCU][j / NCU][k];
        tmp[k % 16] += tmp2_j * tmp2_j;
        for (int p = 0; p < NCU; p++) {
        #pragma HLS unroll
            dataj[p][k] = tmp2_j;
        }
    }

    Loop_add_1:
    for (int j = 0; j < 8; j++) {
    #pragma HLS pipeline
        tmp1[j] = tmp[j] + tmp[j + 8];
    }

    Loop_add_2:
    for (int j = 0; j < 4; j++) {
    #pragma HLS pipeline
        tmp2[j] = tmp1[j] + tmp1[j + 4];
    }

    Loop_add_3:
    for (int j = 0; j < 2; j++) {
    #pragma HLS pipeline
        tmp3[j] = tmp2[j] + tmp2[j + 2];
    }

    tmp3_j = tmp3[0] + tmp3[1];
    tmp1_j = hls::sqrt(dataA[j % NCU][j / NCU][j] - tmp3_j);
    dataA[j % NCU][j / NCU][j] = tmp1_j;
```



Given col j, **pipelined** inner-product its j-1 elements



Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
133	659	1.330 us	6.590 us	133	659	none

Detail

Instance

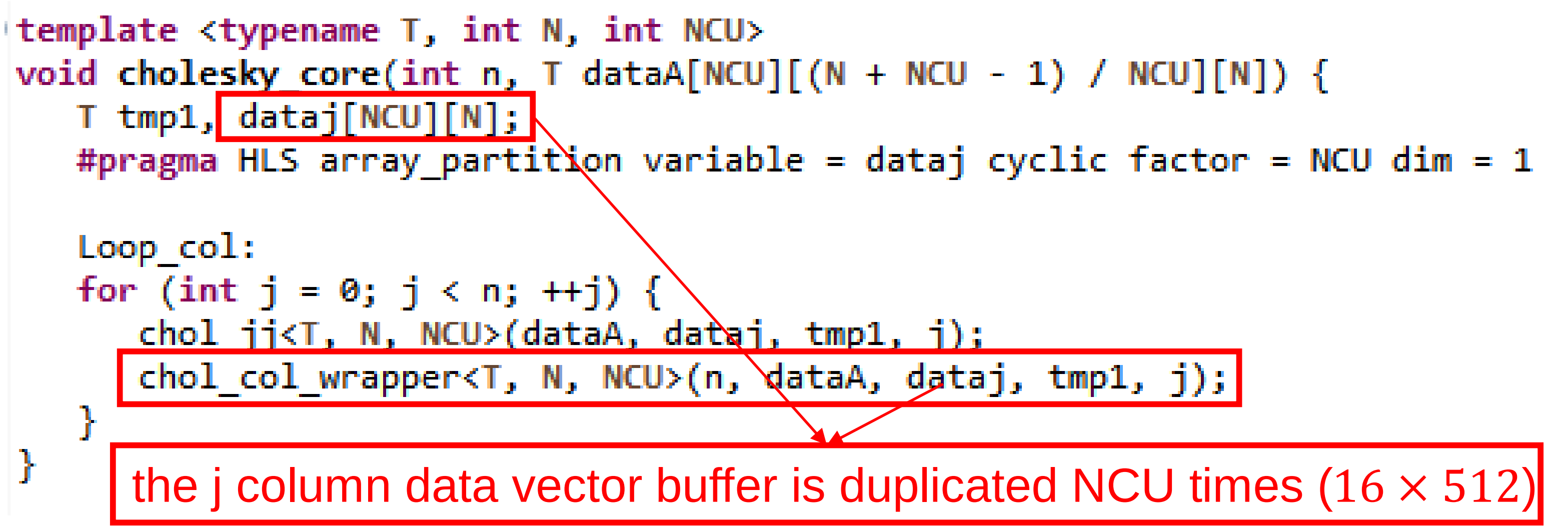
Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	16	16	1	1	1	16	yes
- Loop_vec_mul_jj	0	526	17	1	1	0 ~ 511	yes
- Loop_add_1	16	16	10	1	1	8	yes
- Loop_add_2	11	11	9	1	1	4	yes
- Loop_add_3	9	9	9	1	1	2	yes

Pipelined with II=1, Iter. Lat.= 17

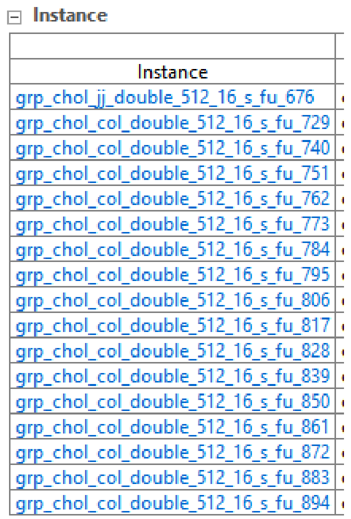
Buffer & Adder Tree to reduce II

chol_col_wrapper unit

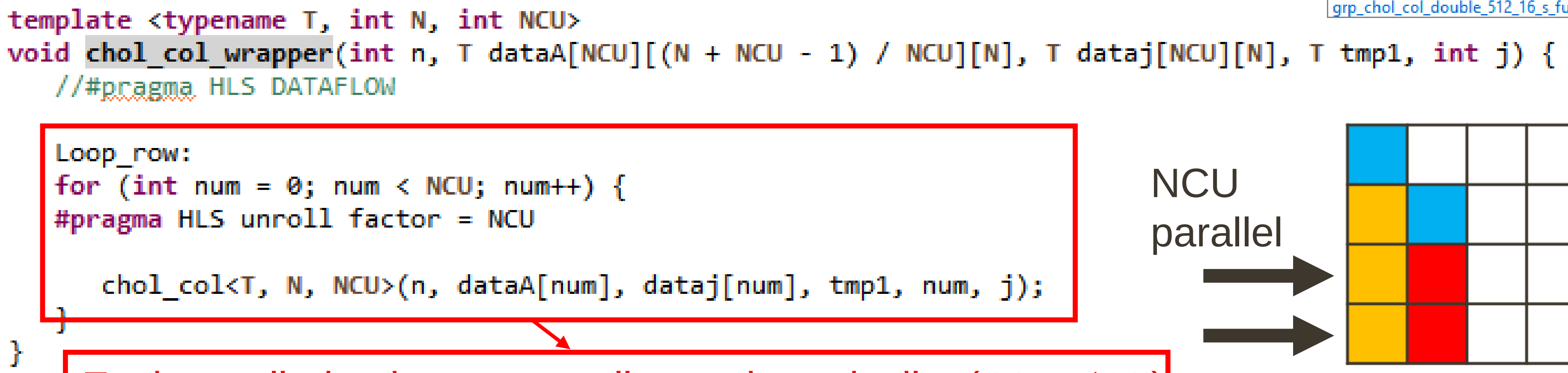


```
template <typename T, int N, int NCU>
void cholesky_core(int n, T dataA[NCU][(N + NCU - 1) / NCU][N]) {
    T tmp1, dataj[NCU][N];
    #pragma HLS array_partition variable = dataj cyclic factor = NCU dim = 1

    Loop_col:
    for (int j = 0; j < n; ++j) {
        chol_jj<T, N, NCU>(dataA, dataj, tmp1, j);
        chol_col_wrapper<T, N, NCU>(n, dataA, dataj, tmp1, j);
    }
}
```

the j column data vector buffer is duplicated NCU times (16 × 512)

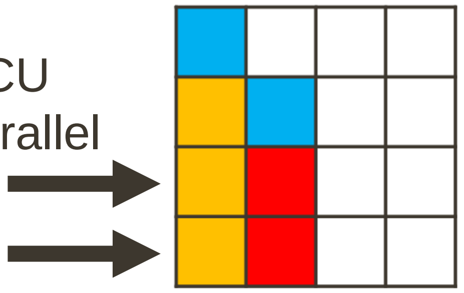


Instance
grp_chol_jj_double_512_16_s_fu_676
grp_chol_col_double_512_16_s_fu_729
grp_chol_col_double_512_16_s_fu_740
grp_chol_col_double_512_16_s_fu_751
grp_chol_col_double_512_16_s_fu_762
grp_chol_col_double_512_16_s_fu_773
grp_chol_col_double_512_16_s_fu_784
grp_chol_col_double_512_16_s_fu_795
grp_chol_col_double_512_16_s_fu_806
grp_chol_col_double_512_16_s_fu_817
grp_chol_col_double_512_16_s_fu_828
grp_chol_col_double_512_16_s_fu_839
grp_chol_col_double_512_16_s_fu_850
grp_chol_col_double_512_16_s_fu_861
grp_chol_col_double_512_16_s_fu_872
grp_chol_col_double_512_16_s_fu_883
grp_chol_col_double_512_16_s_fu_894



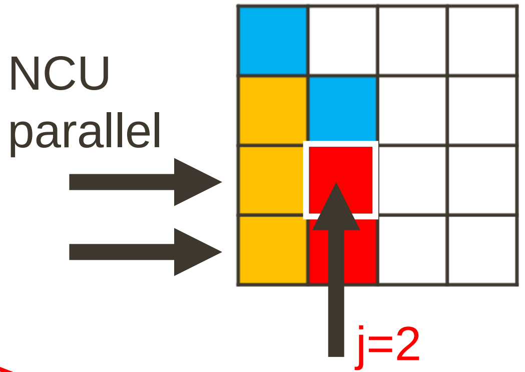
```
template <typename T, int N, int NCU>
void chol_col_wrapper(int n, T dataA[NCU][(N + NCU - 1) / NCU][N], T dataj[NCU][N], T tmp1, int j) {
    //#pragma HLS DATAFLOW

    Loop_row:
    for (int num = 0; num < NCU; num++) {
    #pragma HLS unroll factor = NCU

        chol_col<T, N, NCU>(n, dataA[num], dataj[num], tmp1, num, j);
    }
}
```

NCU parallel



Each unrolled unit gets one allocated matrix tile (32 × 512)

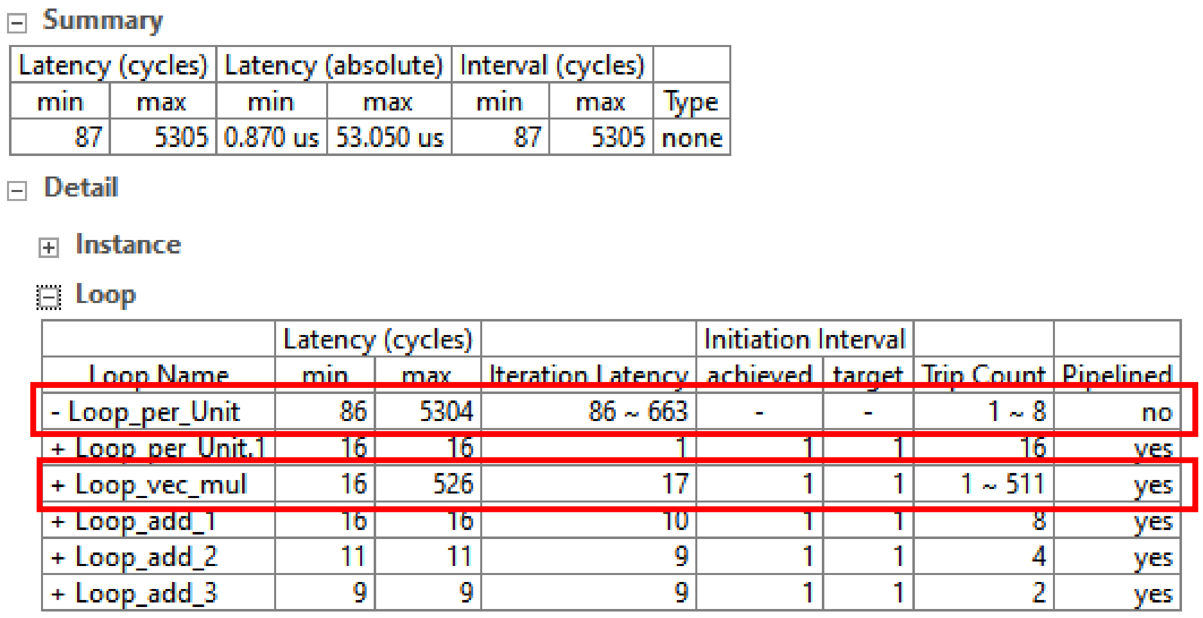
chol_col distributed



```
void chol_col(int n, T dataA[(N + NCU - 1) / NCU][N], T dataj[N], T tmp1_i, int num,
Loop_per_Unit:
for (int p = (j + 1) / NCU; p < (n + NCU - 1) / NCU; p++) {
  #pragma HLS loop_tripcount min = 1 max = 8
   T tmp_i[16] = {0}, tmp3_i, tmp1[8], tmp2[4], tmp3[2];
  #pragma HLS resource variable = tmp_i core = RAM_2P_LUTRAM

  Loop_vec_mul:
  for (int k = 0; k < j; k++) {
  #pragma HLS loop_tripcount min = 1 max = N
  #pragma HLS pipeline
  #pragma HLS dependence variable = tmp_i inter false
  #pragma HLS dependence variable = dataA inter false
     tmp_i[k % 16] += dataA[p][k] * dataj[k];
  }

  Loop_add_1:
  for (int j = 0; j < 8; j++) {
     #pragma HLS pipeline
     tmp1[j] = tmp_i[j] + tmp_i[j + 8];
  }

  Loop_add_2:
  for (int j = 0; j < 4; j++) {
  #pragma HLS pipeline
     tmp2[j] = tmp1[j] + tmp1[j + 4];
  }

  Loop_add_3:
  for (int j = 0; j < 2; j++) {
  #pragma HLS pipeline
     tmp3[j] = tmp2[j] + tmp2[j + 2];
  }

  tmp3_i = tmp3[0] + tmp3[1];

  if (p * NCU + num > j) dataA[p][j] = (dataA[p][j] - tmp3_i) / tmp1_i;
```

Given col j, for the rows p in the given tile

Pipelined inner-product j-1 elements
With i, j column (data[p], dataj)



Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
87	5305	0.870 us	53.050 us	87	5305	none

Detail

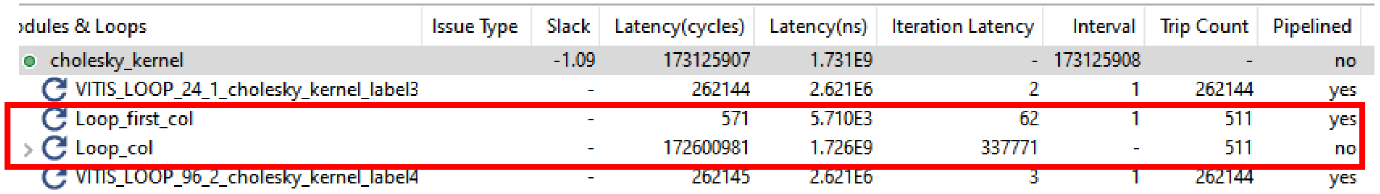
Instance

Loop

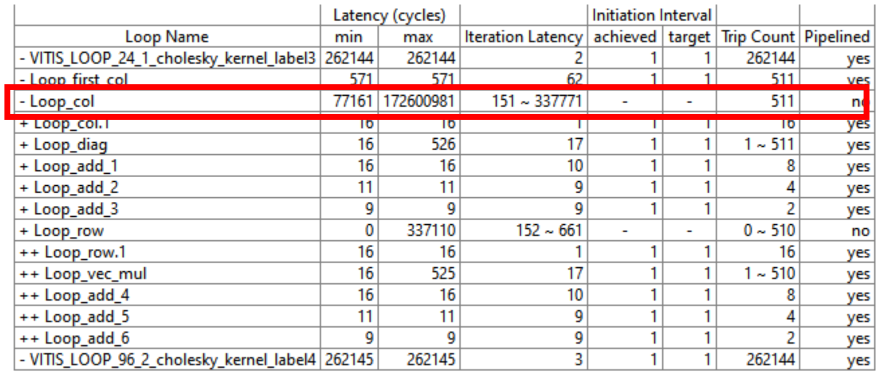
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop_per_Unit	86	5304	86 ~ 663	-	-	1 ~ 8	no
+ Loop_per_Unit.1	16	16	1	1	1	16	yes
+ Loop_vec_mul	16	526	17	1	1	1 ~ 511	yes
+ Loop_add_1	16	16	10	1	1	8	yes
+ Loop_add_2	11	11	9	1	1	4	yes
+ Loop_add_3	9	9	9	1	1	2	yes

Compare Report (512x512)

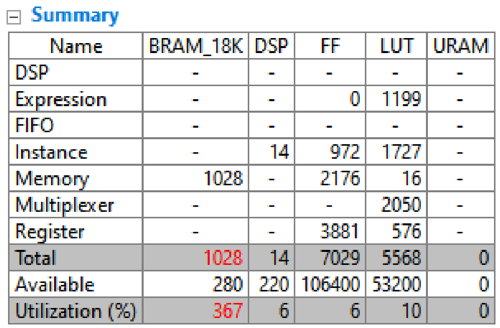
- Pipeline II=1 (max 172600981, max 337771 per col)



dules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined
cholesky_kernel		-1.09	173125907	1.731E9	-	173125908	-	no
VITIS_LOOP_24_1_cholesky_kernel_label3	-		262144	2.621E6	2	1	262144	yes
Loop_first_col	-		571	5.710E3	62	1	511	yes
Loop_col	-		172600981	1.726E9	337771	-	511	no
VITIS_LOOP_96_2_cholesky_kernel_label4	-		262145	2.621E6	3	1	262144	yes



	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- VITIS_LOOP_24_1_cholesky_kernel_label3	262144	262144	2	1	1	262144	yes
- Loop_first_col	571	571	62	1	1	511	yes
- Loop_col	77161	172600981	151 ~ 337771	-	-	511	n
+ Loop_col.1	16	16	1	1	1	16	yes
+ Loop_diag	16	526	17	1	1	1 ~ 511	yes
+ Loop_add_1	16	16	10	1	1	8	yes
+ Loop_add_2	11	11	9	1	1	4	yes
+ Loop_add_3	9	9	9	1	1	2	yes
+ Loop_row	0	337110	152 ~ 661	-	-	0 ~ 510	no
++ Loop_row.1	16	16	1	1	1	16	yes
++ Loop_vec_mul	16	525	17	1	1	1 ~ 510	yes
++ Loop_add_4	16	16	10	1	1	8	yes
++ Loop_add_5	11	11	9	1	1	4	yes
++ Loop_add_6	9	9	9	1	1	2	yes
- VITIS_LOOP_96_2_cholesky_kernel_label4	262145	262145	3	1	1	262144	yes

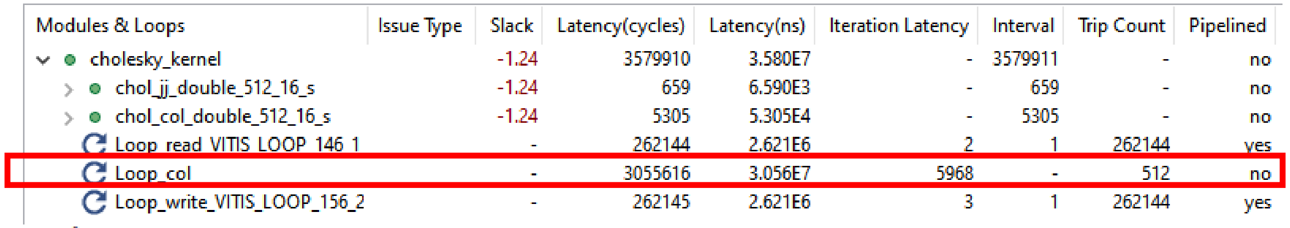


Summary

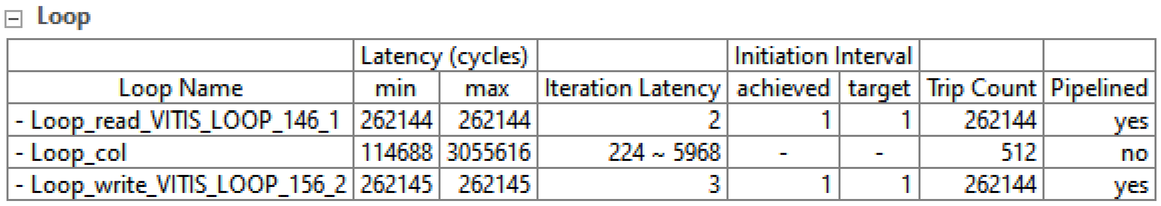
Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1199	-
FIFO	-	-	-	-	-
Instance	-	14	972	1727	-
Memory	1028	-	2176	16	-
Multiplexer	-	-	-	2050	-
Register	-	-	3881	576	-
Total	1028	14	7029	5568	0
Available	280	220	106400	53200	0
Utilization (%)	367	6	6	10	0

x56

- Unroll factor = 16, (max 3055616, max 5968 per col)

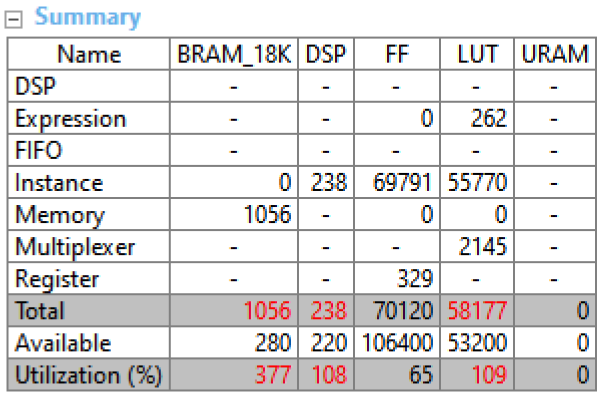


Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined
cholesky_kernel		-1.24	3579910	3.580E7	-	3579911	-	no
chol_jj_double_512_16_s		-1.24	659	6.590E3	-	659	-	no
chol_col_double_512_16_s		-1.24	5305	5.305E4	-	5305	-	no
Loop_read_VITIS_LOOP_146_1	-		262144	2.621E6	2	1	262144	yes
Loop_col	-		3055616	3.056E7	5968	-	512	no
Loop_write_VITIS_LOOP_156_2	-		262145	2.621E6	3	1	262144	yes



Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop_read_VITIS_LOOP_146_1	262144	262144	2	1	1	262144	yes
- Loop_col	114688	3055616	224 ~ 5968	-	-	512	no
- Loop_write_VITIS_LOOP_156_2	262145	262145	3	1	1	262144	yes

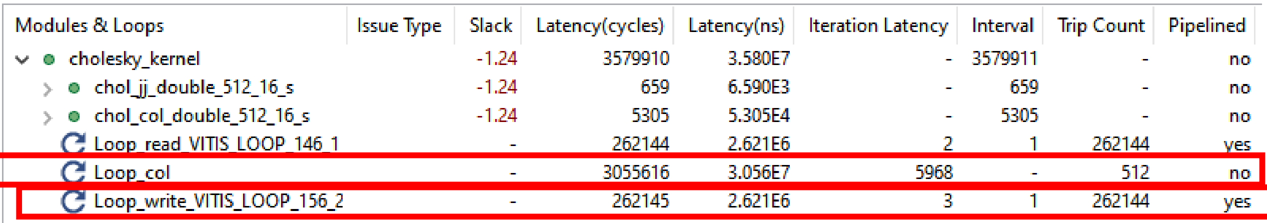


Summary

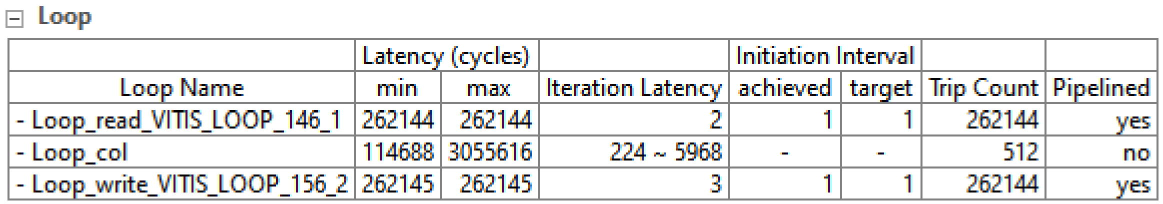
Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	262	-
FIFO	-	-	-	-	-
Instance	0	238	69791	55770	-
Memory	1056	-	0	0	-
Multiplexer	-	-	-	2145	-
Register	-	-	329	-	-
Total	1056	238	70120	58177	0
Available	280	220	106400	53200	0
Utilization (%)	377	108	65	109	0

4. Datatype

- Unroll factor = 16, (max 3055616, max 5968 per col)

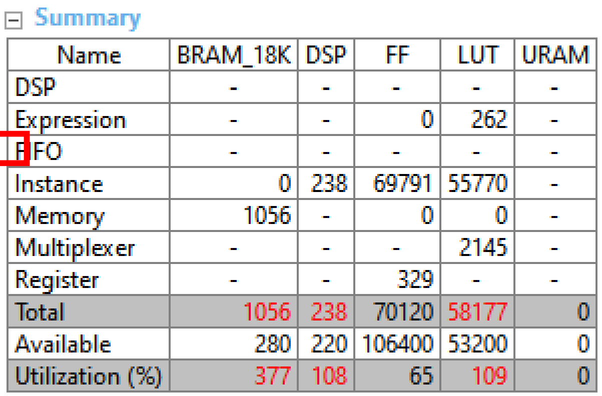


Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined
cholesky_kernel		-1.24	3579910	3.580E7	-	3579911	-	no
chol_jj_double_512_16_s		-1.24	659	6.590E3	-	659	-	no
chol_col_double_512_16_s		-1.24	5305	5.305E4	-	5305	-	no
Loop_read_VITIS_LOOP_146_1		-	262144	2.621E6	2	1	262144	yes
Loop_col		-	3055616	3.056E7	5968	-	512	no
Loop_write_VITIS_LOOP_156_2		-	262145	2.621E6	3	1	262144	yes



Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop_read_VITIS_LOOP_146_1	262144	262144	2	1	1	262144	yes
- Loop_col	114688	3055616	224 ~ 5968	-	-	512	no
- Loop_write_VITIS_LOOP_156_2	262145	262145	3	1	1	262144	yes

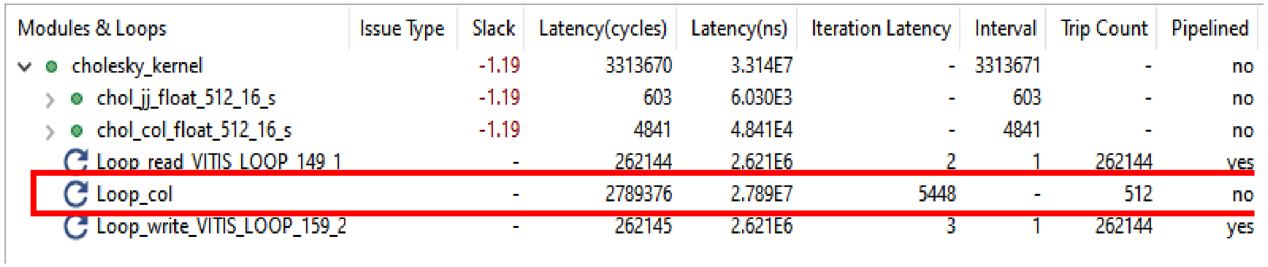


Summary

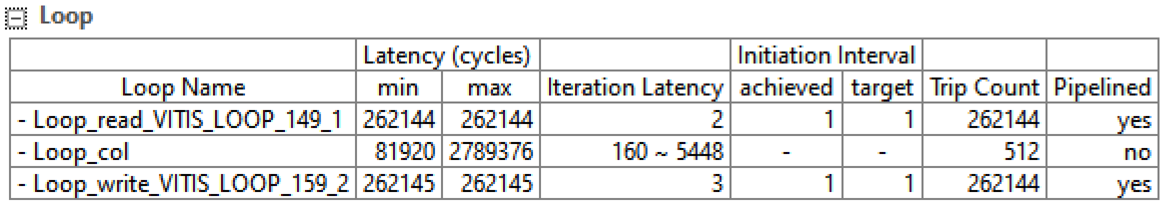
Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	262	-
FIFO	-	-	-	-	-
Instance	0	238	69791	55770	-
Memory	1056	-	0	0	-
Multiplexer	-	-	-	2145	-
Register	-	-	329	-	-
Total	1056	238	70120	58177	0
Available	280	220	106400	53200	0
Utilization (%)	377	108	65	109	0

- Float datatype, (max 2789376, max 5448 per col)

Resource Utilization X0.5

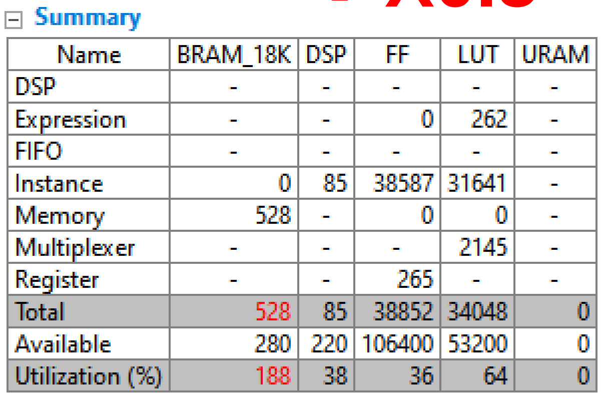


Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined
cholesky_kernel		-1.19	3313670	3.314E7	-	3313671	-	no
chol_jj_float_512_16_s		-1.19	603	6.030E3	-	603	-	no
chol_col_float_512_16_s		-1.19	4841	4.841E4	-	4841	-	no
Loop_read_VITIS_LOOP_149_1		-	262144	2.621E6	2	1	262144	yes
Loop_col		-	2789376	2.789E7	5448	-	512	no
Loop_write_VITIS_LOOP_159_2		-	262145	2.621E6	3	1	262144	yes



Loop

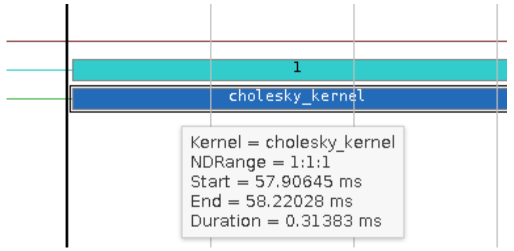
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop_read_VITIS_LOOP_149_1	262144	262144	2	1	1	262144	yes
- Loop_col	81920	2789376	160 ~ 5448	-	-	512	no
- Loop_write_VITIS_LOOP_159_2	262145	262145	3	1	1	262144	yes



Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	262	-
FIFO	-	-	-	-	-
Instance	0	85	38587	31641	-
Memory	528	-	0	0	-
Multiplexer	-	-	-	2145	-
Register	-	-	265	-	-
Total	528	85	38852	34048	0
Available	280	220	106400	53200	0
Utilization (%)	188	38	36	64	0

Vitis System



Kernel = cholesky_kernel
NDRange = 1:1:1
Start = 57.90645 ms
End = 58.22028 ms
Duration = 0.31383 ms

Vitis System

CPU

(ms)	16x16	512x512
Total	0.007	**15.6**

U50 System (Pipelined II=6)

(ms)	16x16	512x512
Total	112.8	956
Kernel	0.313	655
Memory Write	0.124	0.527
Memory Read	0.057	0.527

U50 System (Pipelined II=1)

(ms)	16x16	512x512
Total	94.4	2880
Kernel	0.21	156
Memory Write	0.096	0.511
Memory Read	0.021	0.502

U50 System (Unroll factor=16)

(ms)	16x16	512x512
Total		2727
Kernel		12.4
Memory Write		0.318
Memory Read		0.509

U50 System (Unroll factor=16 + float)

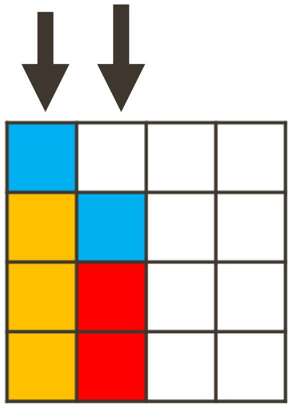
(ms)	16x16	512x512
Total		2660
Kernel		**10.7**
Memory Write		0.184
Memory Read		0.231

Appendix

4. Code Refactor – Dataflow

- 4.1. Dataflow @ inner product & adder tree
- 4.2. Dataflow @ column -> Pipeline @ column
- 4.3. Dataflow @ read_loop & cholesky_core & write_loop

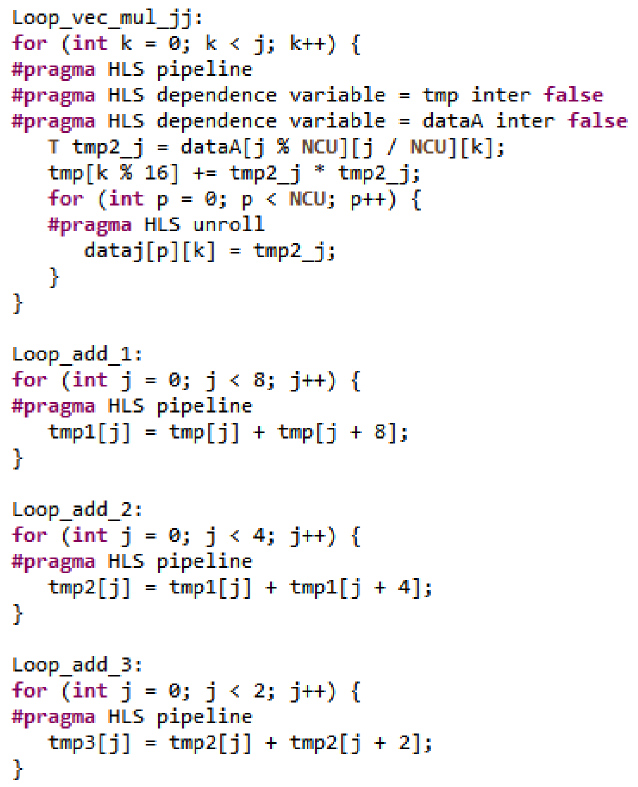
```
Loop_vec_mul_jj:
for (int k = 0; k < j; k++) {
#pragma HLS pipeline
#pragma HLS dependence variable = tmp inter false
#pragma HLS dependence variable = dataA inter false
    T tmp2_j = dataA[j % NCU][j / NCU][k];
    tmp[k % 16] += tmp2_j * tmp2_j;
    for (int p = 0; p < NCU; p++) {
    #pragma HLS unroll
        dataj[p][k] = tmp2_j;
    }
}

Loop_add_1:
for (int j = 0; j < 8; j++) {
#pragma HLS pipeline
    tmp1[j] = tmp[j] + tmp[j + 8];
}

Loop_add_2:
for (int j = 0; j < 4; j++) {
#pragma HLS pipeline
    tmp2[j] = tmp1[j] + tmp1[j + 4];
}

Loop_add_3:
for (int j = 0; j < 2; j++) {
#pragma HLS pipeline
    tmp3[j] = tmp2[j] + tmp2[j + 2];
}
```



```
Loop_read:
for (int r = 0; r < m; r++) {
    for (int c = 0; c < m; c++) {
    #pragma HLS pipeline
    matA[r % NCU][r / NCU][c] = A[r * lda + c];
    }
}

internal::cholesky_core<T, NMAX, NCU>(m, matA);

Loop_write:
for (int r = 0; r < m; r++) {
    for (int c = 0; c < m; c++) {
    #pragma HLS pipeline
        A[r * lda + c] = matA[r % NCU][r / NCU][c];
    }
}
```

4.1. Dataflow @ inner product & adder tree

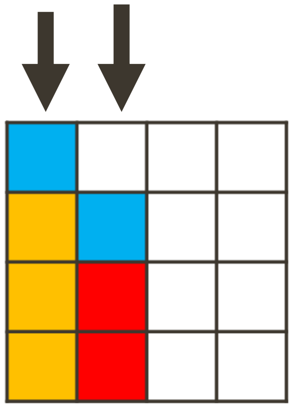
- Do not really help, since the input of adder tree has to wait for ***all*** previous outputs.



```
Loop_vec_mul_jj:
for (int k = 0; k < j; k++) {
#pragma HLS pipeline
#pragma HLS dependence variable = tmp inter false
#pragma HLS dependence variable = dataA inter false
   T tmp2_j = dataA[j % NCU][j / NCU][k];
   tmp[k % 16] += tmp2_j * tmp2_j;
   for (int p = 0; p < NCU; p++) {
   #pragma HLS unroll
      dataj[p][k] = tmp2_j;
   }
}

Loop_add_1:
for (int j = 0; j < 8; j++) {
#pragma HLS pipeline
   tmp1[j] = tmp[j] + tmp[j + 8];
}

Loop_add_2:
for (int j = 0; j < 4; j++) {
#pragma HLS pipeline
   tmp2[j] = tmp1[j] + tmp1[j + 4];
}

Loop_add_3:
for (int j = 0; j < 2; j++) {
#pragma HLS pipeline
   tmp3[j] = tmp2[j] + tmp2[j + 2];
}
```

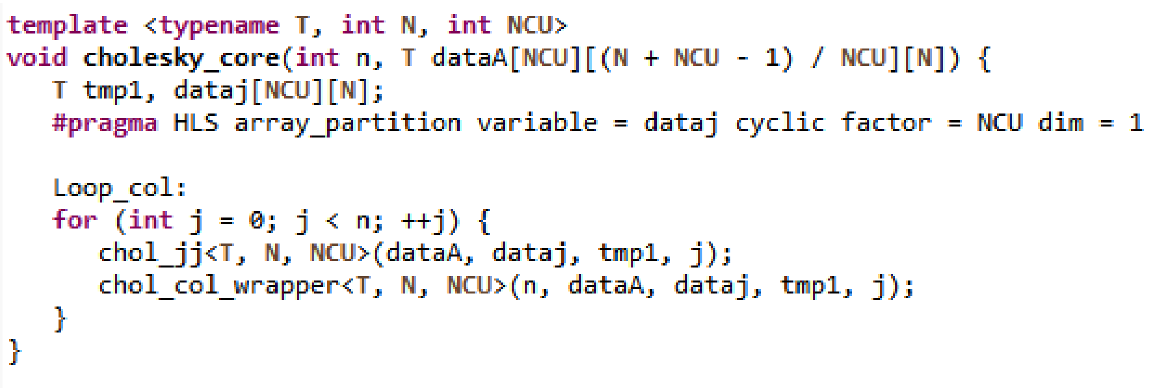
4.2. Dataflow @ column -> Pipeline @ column

- Cannot be pipelined -> no other computing unit available

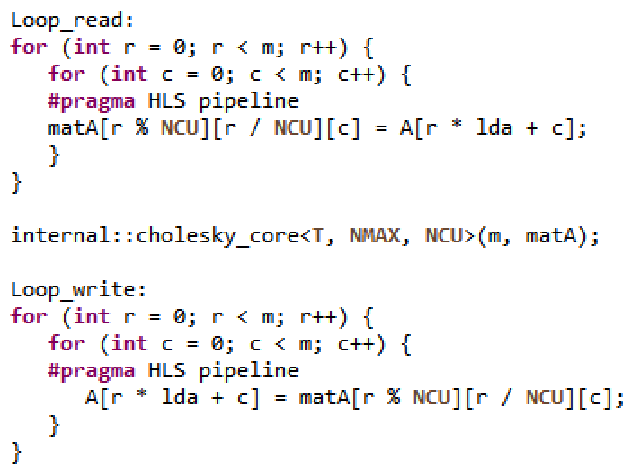


4.3. Dataflow @ read_loop & cholesky_core & write_loop

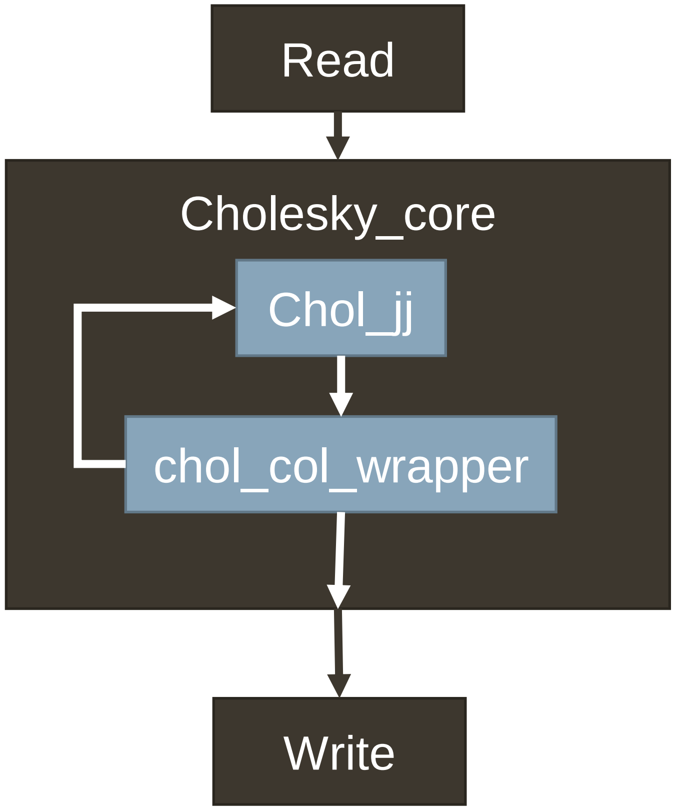
- Not trivial, since there exists feed back loops



```
template <typename T, int N, int NCU>
void cholesky_core(int n, T dataA[NCU][(N + NCU - 1) / NCU][N]) {
   T tmp1, dataj[NCU][N];
   #pragma HLS array_partition variable = dataj cyclic factor = NCU dim = 1

   Loop_col:
   for (int j = 0; j < n; ++j) {
      chol_jj<T, N, NCU>(dataA, dataj, tmp1, j);
      chol_col_wrapper<T, N, NCU>(n, dataA, dataj, tmp1, j);
   }
}
```



```
Loop_read:
for (int r = 0; r < m; r++) {
   for (int c = 0; c < m; c++) {
   #pragma HLS pipeline
   matA[r % NCU][r / NCU][c] = A[r * lda + c];
   }
}

internal::cholesky_core<T, NMAX, NCU>(m, matA);

Loop_write:
for (int r = 0; r < m; r++) {
   for (int c = 0; c < m; c++) {
   #pragma HLS pipeline
      A[r * lda + c] = matA[r % NCU][r / NCU][c];
   }
}
```


Read
Cholesky_core
Chol_jj
chol_col_wrapper
Write